Cyberscope

Audit Report

Jesus 2.0

July 2023

Network	ETH
Address	0xc182266788Ec87E4f66679E78D42B6BdAB878F3E
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	RSW	Redundant Storage Writes	Unresolved
●	CR	Code Repetition	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	L03	Redundant Statements	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved

Table of Contents

Review

Contract Name	Jesus
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://etherscan.io/address/0xc182266788ec87e4f66679e78d42b6bdab878f3e
Address	0xc182266788ec87e4f66679e78d42b6bdab878f3e
Network	ETH
Symbol	$Jesus 2.0.
Decimals	18
Total Supply	1,000,000,000,000

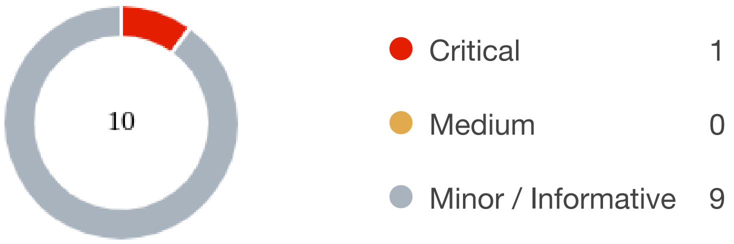
Audit Updates

Initial Audit	09 Jul 2023

Source Files

Filename	SHA256
Jesus.sol	521e65c994f71bd0a66506c7f028a8298add9d8cd9baf54ef231d30c79bfdf65

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	1	0	0	0
● Medium	0	0	0	0
● Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Jesus.sol#L522
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
if (from != owner() && to != owner() && to != address(0) && to
!= address(0xdead)){
    if(!tradingActive){
      require(_isExcludedFromFees[from] ||
_isExcludedFromFees[to]"Trading is not active.");
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Jesus.sol#L450,479
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function updateSwapTokensAtAmount(uint256 newAmount) external
onlyOwner {
    require(newAmount >= totalSupply() * 1 / 100000, "Swap
amount cannot be lower than 0.001% total supply.");
    require(newAmount <= totalSupply() * 3 / 100, "Swap amount
cannot be higher than 3% total supply.");
      swapTokensAtAmount = newAmount * (10**18);
}
```

```
function updateFees(uint256 _buyMarketingFee, uint256
_buyLiquidityFee, uint256 _buyBurnFee, uint256
_sellMarketingFee, uint256 _sellLiquidityFee, uint256
_sellBurnFee) external onlyOwner {
    buyMarketingFee = _buyMarketingFee;
    buyLiquidityFee = _buyLiquidityFee;
    buyBurnFee = _buyBurnFee;
    sellMarketingFee = _sellMarketingFee;
    sellLiquidityFee = _sellLiquidityFee;
    sellBurnFee = _sellBurnFee;
    sellTotalFees = sellMarketingFee + sellLiquidityFee +
sellBurnFee;
    buyTotalFees = buyMarketingFee + buyLiquidityFee +
buyBurnFee;
    require(buyTotalFees <= 5,"Total buy fees cannot be greater
than 5%");
    require(sellTotalFees <= 5,"Total sell fees cannot be
greater than 5%");
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Jesus.sol#L569,576
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
// on sell
    if (automatedMarketMakerPairs[to] && sellTotalFees > 0){
        fees = amount * (sellTotalFees)/(100);
        tokensForLiquidity += fees * sellLiquidityFee /
sellTotalFees;
        tokensForMarketing += fees * sellMarketingFee /
sellTotalFees;
        tokensForBurn += fees * sellBurnFee / sellTotalFees;
    }
// on buy
    else if(automatedMarketMakerPairs[from] && buyTotalFees >
0) {
        fees = amount * (buyTotalFees) / (100);
        tokensForLiquidity += fees * buyLiquidityFee /
buyTotalFees;
        tokensForMarketing += fees * buyMarketingFee /
buyTotalFees;
        tokensForBurn += fees * buyBurnFee / buyTotalFees;
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

CR - Code Repetition

Criticality	Minor / Informative
Location	Jesus.sol#L568
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
fees = amount * (sellTotalFees)/(100);
tokensForLiquidity += fees * sellLiquidityFee / sellTotalFees;
tokensForMarketing += fees * sellMarketingFee / sellTotalFees;
tokensForBurn += fees * sellBurnFee / sellTotalFees;
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Jesus.sol#L493,510
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract updates the state of excluded addresses even if their current state is the same as the the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function updateSwapEnabled(bool enabled) external onlyOwner(){
    swapEnabled = enabled;
}

function excludeFromFees(address account, bool excluded) public
onlyOwner {
    _isExcludedFromFees[account] = excluded;
    emit ExcludeFromFees(account, excluded);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

L03 - Redundant Statements

Criticality	Minor / Informative
Location	Jesus.sol#L401,406
Status	Unresolved

Description

Redundant statements are statements that are unnecessary or have no effect on the contract's behavior. These can include declarations of variables or functions that are not used, or assignments to variables that are never used.

As a result, it can make the contract's code harder to read and maintain, and can also increase the contract's size and gas consumption, potentially making it more expensive to deploy and execute.

```
buyLiquidityFee;
sellLiquidityFee;
```

Recommendation

To avoid redundant statements, it's important to carefully review the contract's code and remove any statements that are unnecessary or not used. This can help to improve the clarity and efficiency of the contract's code.

By removing unnecessary or redundant statements from the contract's code, the clarity and efficiency of the contract will be improved. Additionally, the size and gas consumption will be reduced.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Jesus.sol#L269,343,479,497,504
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);
mapping (address => bool) public
_isExcludedMaxTransactionAmount;
function updateFees(uint256 _buyMarketingFee, uint256
_buyLiquidityFee, uint256 _buyBurnFee, uint256
_sellMarketingFee, uint256 _sellLiquidityFee, uint256
_sellBurnFee) external onlyOwner
function transferForeignToken(address _token, address _to)
external onlyOwner returns (bool _sent)
function setMarketingAddress(address _marketingAddress)
external onlyOwner
...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Jesus.sol#L569,570,571,575,576,577,578
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
tokensForLiquidity += fees * sellLiquidityFee / sellTotalFees;
tokensForMarketing += fees * sellMarketingFee / sellTotalFees;
tokensForBurn += fees * sellBurnFee / sellTotalFees;
fees = amount * (buyTotalFees) / (100);
tokensForLiquidity += fees * buyLiquidityFee / buyTotalFees;
tokensForMarketing += fees * buyMarketingFee / buyTotalFees;
tokensForBurn += fees * buyBurnFee / buyTotalFees;
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Jesus.sol#L545
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 tokensForBurn;
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Jesus.sol#L389
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint256 totalSupply = 1000000000000 * 1e18;
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

Functions Analysis

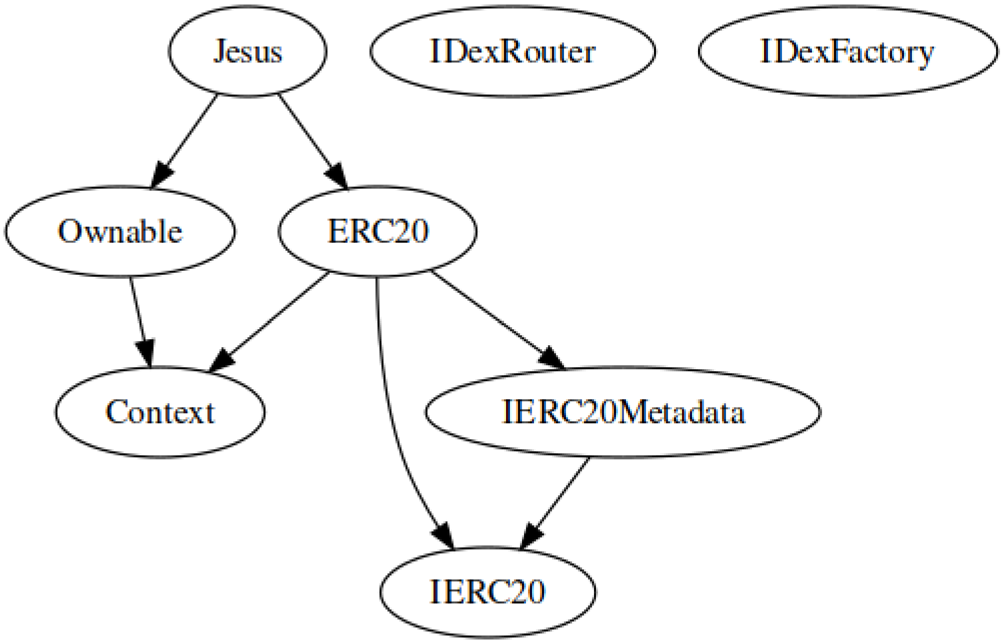
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_createInitialSupply	Internal	✓	
	_approve	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	External	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner

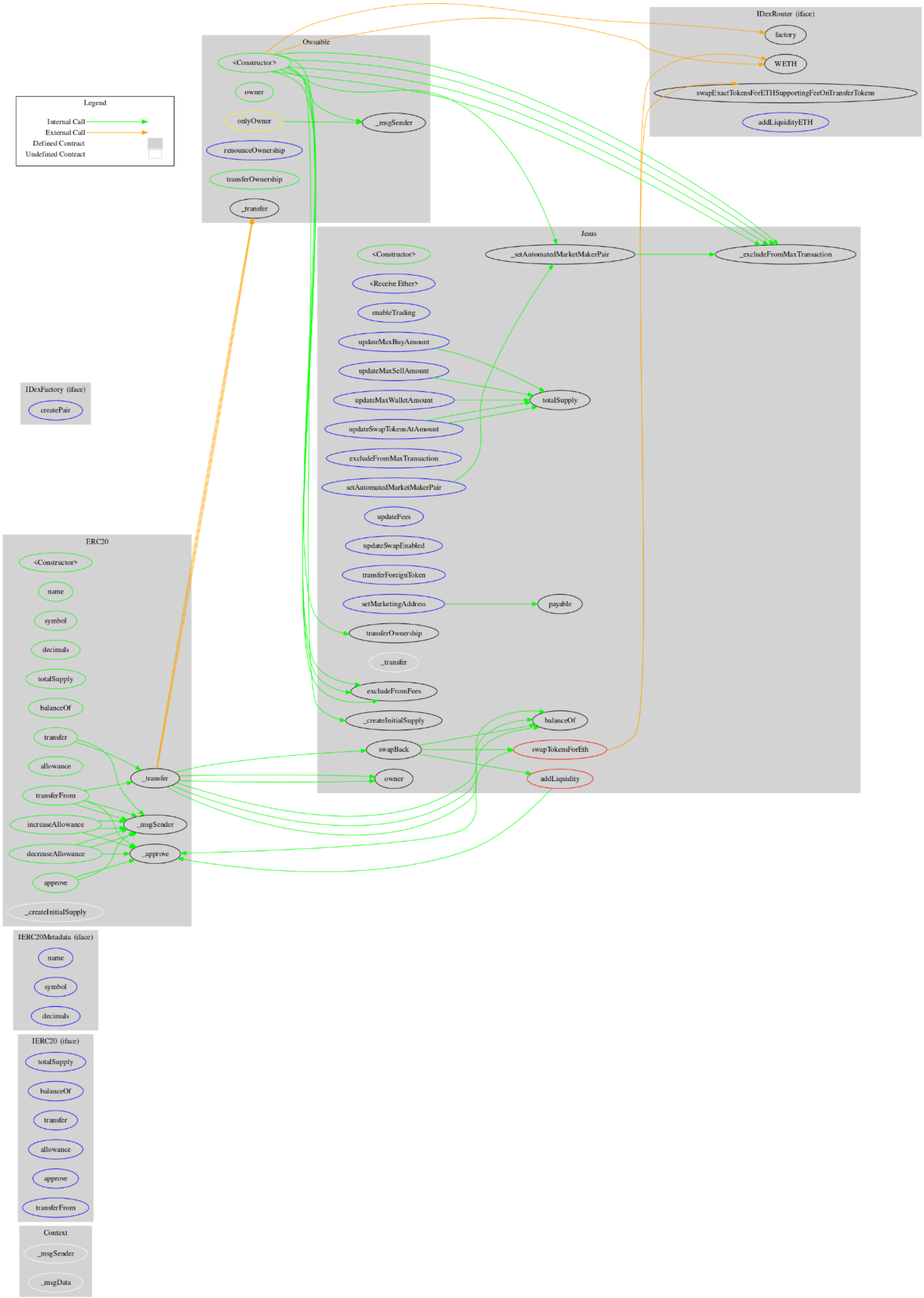
IDexRouter	Interface			
	factory	External		-
	WETH	External		-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	addLiquidityETH	External	Payable	-
IDexFactory	Interface			
	createPair	External	✓	-
Jesus	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	updateMaxBuyAmount	External	✓	onlyOwner
	updateMaxSellAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	_excludeFromMaxTransaction	Private	✓	
	excludeFromMaxTransaction	External	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateFees	External	✓	onlyOwner

	updateSwapEnabled	External	✓	onlyOwner
	transferForeignToken	External	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
<Constructor>
owner
onlyOwner
_msgSender
renounceOwnership
transferOwnership
_transfer
IDexRouter (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
addLiquidityETH
Jesus
<Constructor>
_setAutomatedMarketMakerPair
_excludeFromMaxTransaction
<Receive Ether>
enableTrading
updateMaxBuyAmount
updateMaxSellAmount
updateMaxWalletAmount
totalSupply
updateSwapTokensAtAmount
excludeFromMaxTransaction
setAutomatedMarketMakerPair
updateFees
updateSwapEnabled
transferForeignToken
setMarketingAddress
payable
transferOwnership
_transfer
excludeFromFees
_createInitialSupply
balanceOf
swapBack
swapTokensForEth
owner
addLiquidity
IDexFactory (iface)
createPair
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
_transfer
transferFrom
increaseAllowance
_msgSender
decreaseAllowance
_approve
approve
_createInitialSupply
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData

Summary

Jesus contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract will eliminate all the contract threats. There is also a limit of max 5% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io